Karel Assignment

Eyass Bdair

Divide a given map into 4 equal chambers – take into consideration the special cases such as small maps that can’t be divided into 4 chambers equally and divide them into the maximum possible number of equal chambers (1, 2, or 3) .

The following considerations are taken into account:

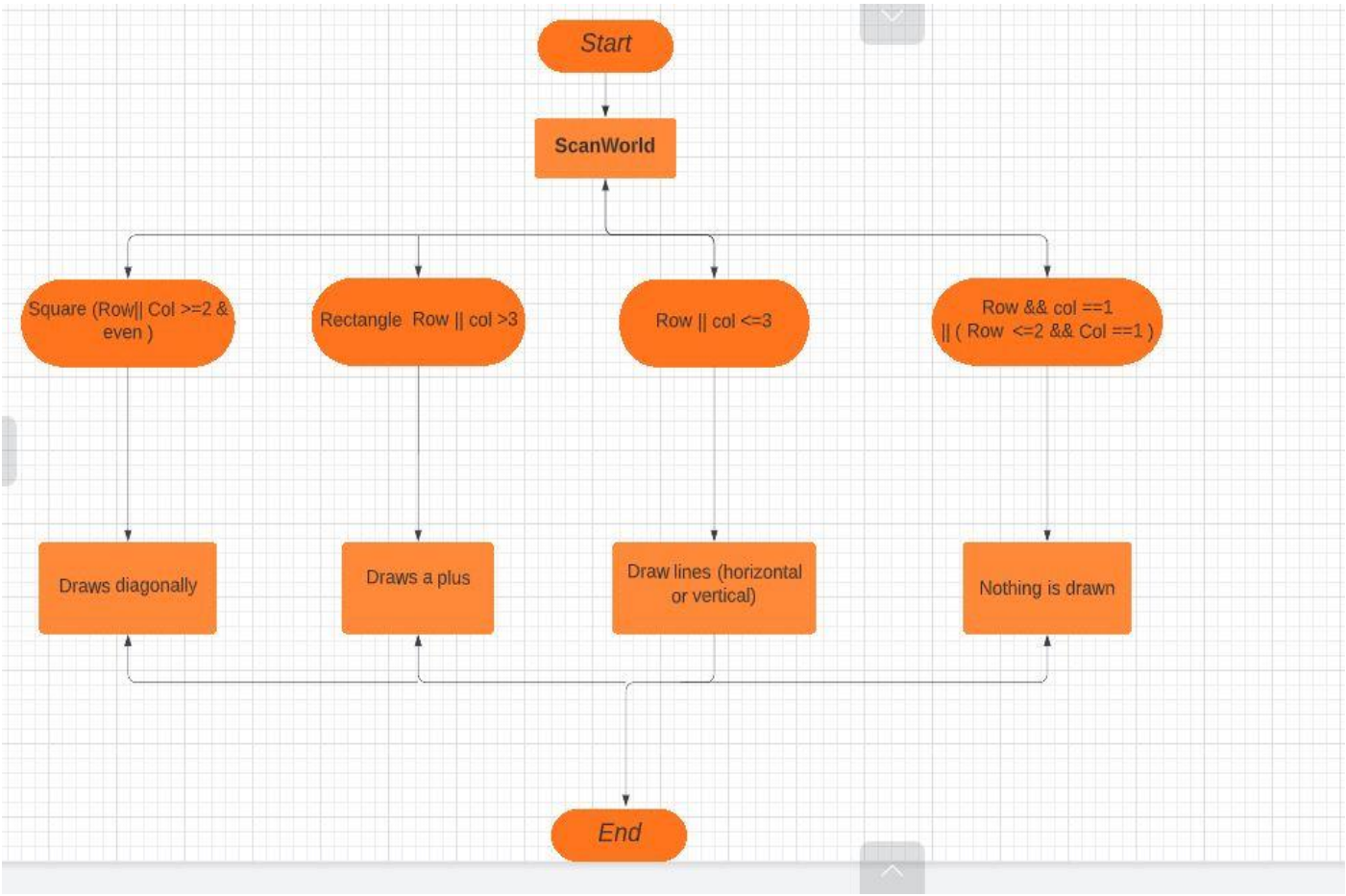
- Karel should achieve his task with the lowest number of moves, add a moves counter to your code and print it while Karel is moving.

- You should minimize the number of lines in your code to the lowest possible number of lines by writing reusable functions.

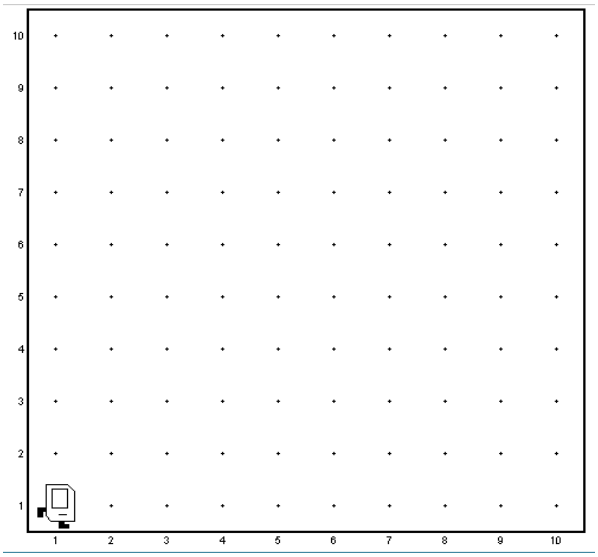
- Use the lowest possible number of beepers to achieve your task

Describe how the assignment was solved :

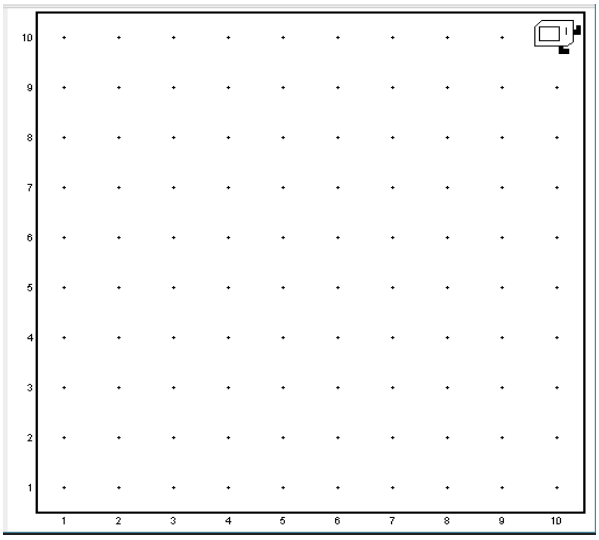
First FlowChart for Karel Assignment :



First, Karel scans the world to calculate the dimensions of the map (length and width) via the ScanWorld function , And then determine the method of drawing whether it is (diameter, vertical, horizontal, plus, or the map cannot be divided)



before Scanning Map



after Scanning Map

```
Row = 10
Column = 10
steps = 19
beepers = 0
```

As you can see, Karel scanned the map, calculated the dimensions, and also printed the dimensions, the number of steps he took, and the number of Beepers he used (Here it did not use any cuts because he did not call any algorithm).

And after scanning the map , Based on the algorithms you wrote, Karel Robot by divides and draws the map according to specific algorithms according to each case, where each case has its own section so that Karel performs the least number of moves and has the least number of Beeper pieces according to the algorithm.

Now I will explain the cases with the solution according to the algorithm of the case :

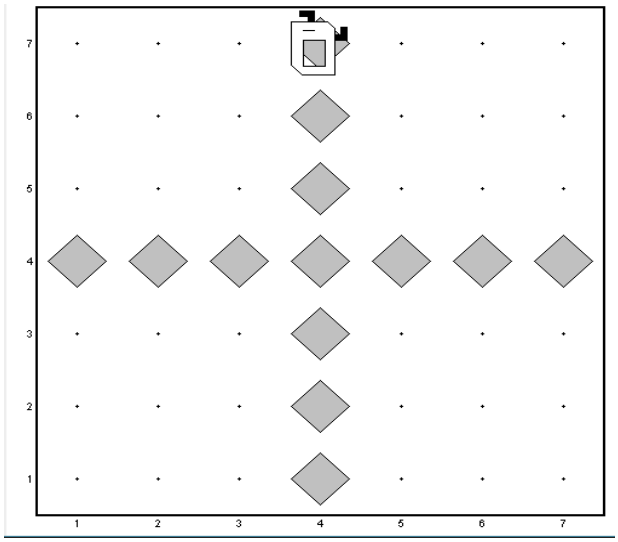
Drow Plus (General Algorithm):

This case includes most cases and is the general algorithm.

After the scanning process, and in the event that the dimensions of the map are (odd and odd or even and odd or even and even provided that the map is not square) and the dimensions are greater than 3, then this case is suitable for the map.

This algorithm divides the dimensions in half and then draws a column in the opposite direction at the midpoint, then Karel moves to the other dimension and does the same way, and if one or both dimensions are even (provided that the map is not square) and draws two lines instead of a line at the midpoint and so on.

Now I will show some cases of this algorithm :

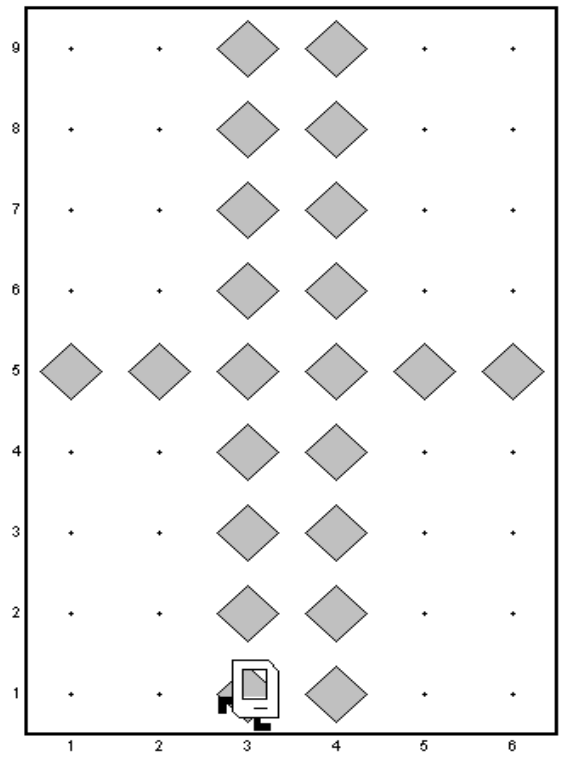


```
Row = 7
Column = 7
steps = 34
beepers = 13
```

In this case the Row is odd equal 7 , and Column is odd equal 7

This case is suitable for this algorithm , Karel used 13 **Beepers** and 34 **Steps** .

Other case :



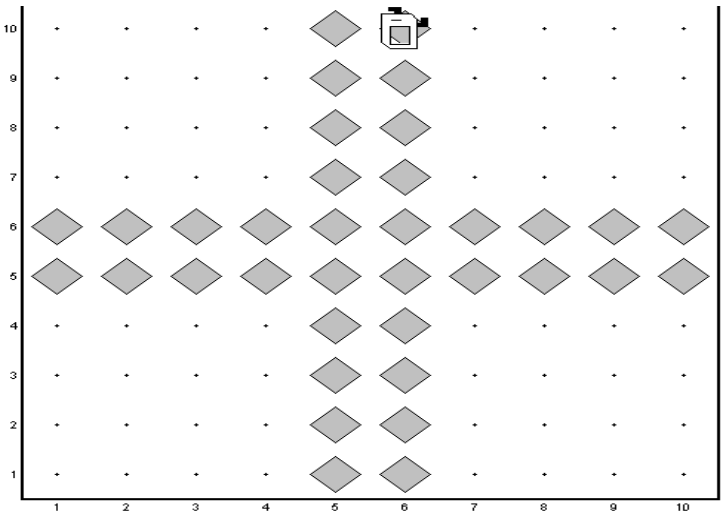

```
Row = 9
Column = 6
steps = 47
beepers = 22
```

In this case the Row is odd equal 9 , and Column is even equal 6

This case is suitable for this algorithm , Karel used 22 **Beepers** and 47 **Steps** .

Draws Diagonally :

When the dimensions of the map are even and equal, the previous algorithm can be used (extra drawing) and at first, I solved it this way, but now after examining all the cases, a solution has been found with the least number of moves and the least amount of Beepers.



```
Row = 10
Column = 10
steps = 71
beepers = 36
```

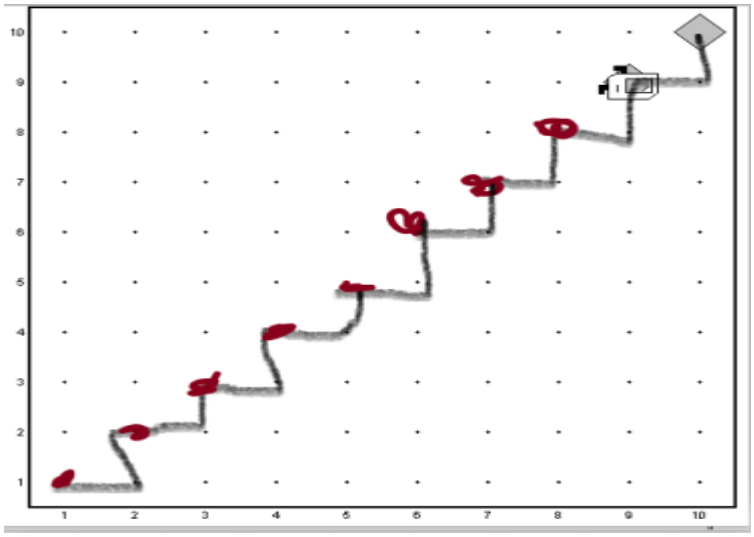
In this case the Row is even equal 10 , and Column is even equal 10

This case is suitable for this algorithm , Karel used 36 **Beepers** and 71 **Steps** .

This is how it was solved by the first algorithm, and now it is solved by another method and algorithm .

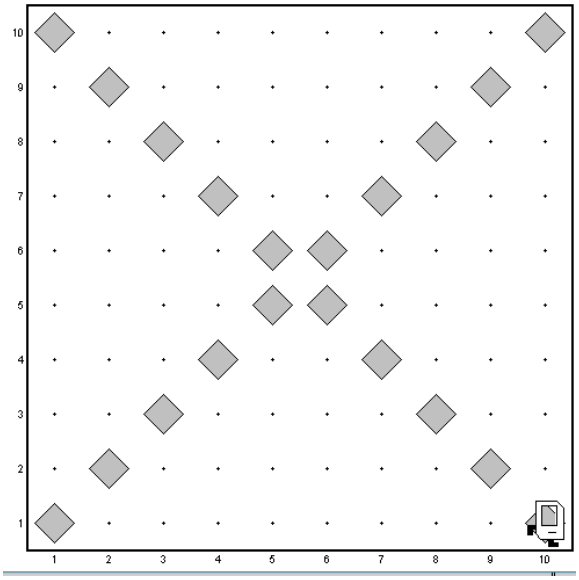
After the map scanning process, if the dimensions of the map are equal and the dimensions are even, that is, it is square in shape, and this algorithm (draw diagonally) is suitable for this map.

This algorithm moves diagonally and places the whistle pieces (the movement is by moving down , then moving to the right , and so on until the far corner), then moving to the second dimension and drawing the oblique line in the same way.



movement method

Now I will show some cases of this algorithm :



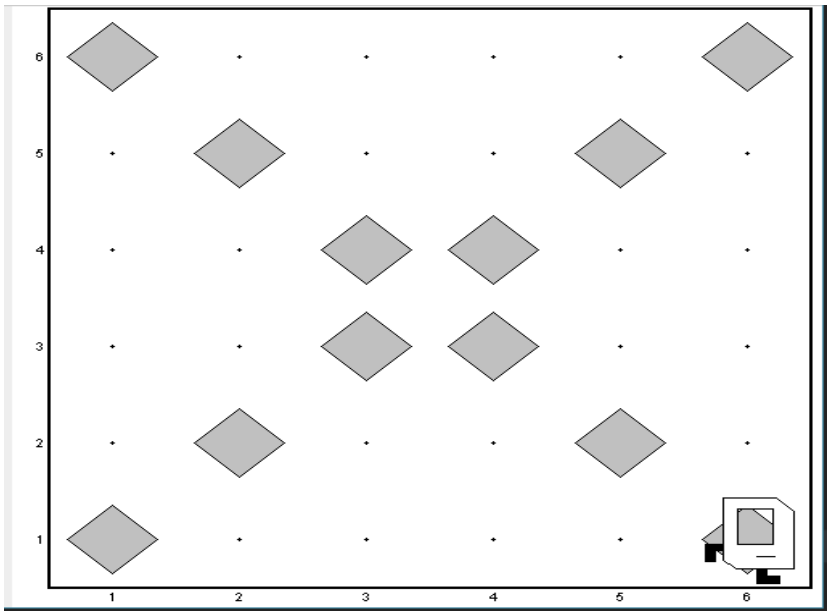
```
Row = 10
Column = 10
steps = 64
beepers = 20
```

In this case the Row is even equal 10 , and Column is even equal 10

This case is suitable for this algorithm , Karel used 20 **Beepers** and

64 **Steps** .

Other case :



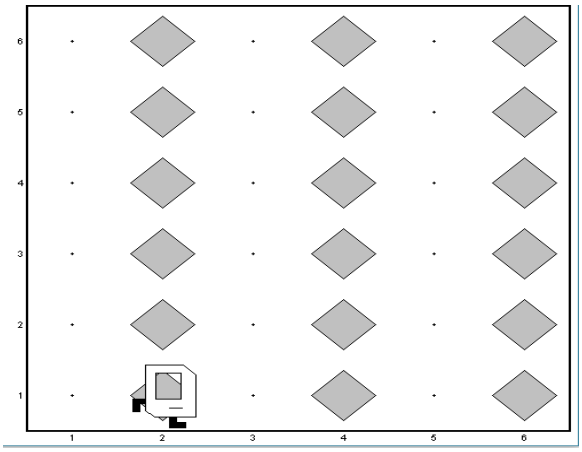
```
Row = 6
Column = 6
steps = 36
beepers = 12
```

In this case the Row is even equal 6 , and Column is odd even 6

This case is suitable for this algorithm , Karel used 12 **Beepers** and

36 **Steps** .

Of course, it can be solved by drawing vertical or horizontal lines, but it will require more moves and cuts than drawing the diagonal. But the map will be divided into 3 parts, and according to the requirements of the question, the maximum number of pieces must be equal to 4 or fewer. As in the following picture



Draw Lines (horizontal or vertical) :

There are special cases where the map cannot be divided into four sections using the previous algorithms, as it is not possible to draw lines from the middle or diagonals because one of them is longer than 2 or less and the other is longer than 2.

There are two ways to solve this case. Both solutions have been written, but the first solution is long in terms of code. I'll explain it quickly, in short by taking the largest dimension and frames given the number of barriers I need to divide them into 4 sections (3 barriers or rows) and then check the number if it is divisible by 4 or not, if it is divisible, good luck and divide quickly, but If it is not divisible, find the first number that is divisible by four by subtracting it once and increasing the number of barriers etc. Then put the additional barriers at the beginning and divide the map by the third barrier

Now explain how the other algorithm works and the code works with it

First, there are special cases when the other dimension is less than 7 (I will talk about it later).

The algorithm works as follows:

1) The algorithm takes the largest dimension and divides it into two halves

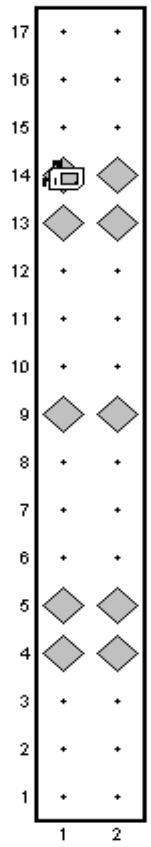
2) If the distance is even, you will put two pieces in the middle, and if it is single, one cat from the Beepers

3) You calculate the first half and divide it into two halves (as in the previous step, depending on whether it is odd or even).

4) You calculate the other half and divide it into two halves, like the previous two steps

Thus, the map is divided into four parts

Now I will show some cases of this algorithm :

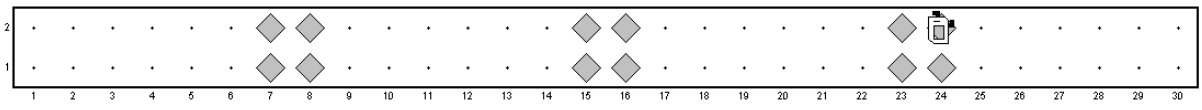



```
Row = 17
Column = 2
steps = 48
beepers = 10
```

In this case the Row is odd equal 17 , and Column is even equal 2

This case is suitable for this algorithm , Karel used 10 **Beepers** and 48 **Steps** .

Other case :

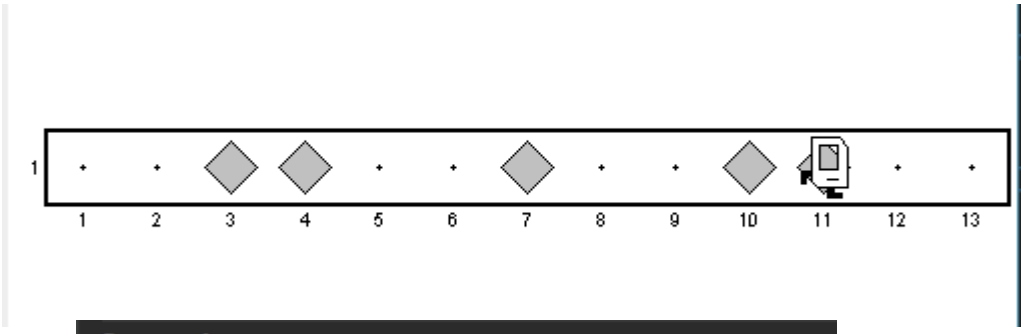



```
Row = 2
Column = 30
steps = 77
beepers = 12
```

In this case the Row is even equal 2 , and Column is even equal 30

This case is suitable for this algorithm , Karel used 12 **Beepers** and 77 **Steps** .

Other case :

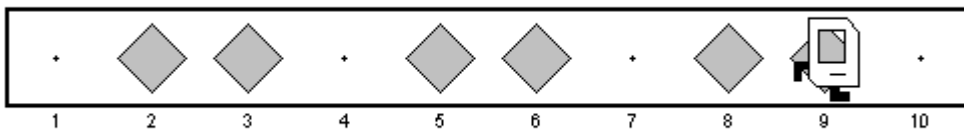



```
Row = 1
Column = 13
steps = 31
beepers = 5
```

In this case the Row is odd equal 1 , and Column is odd equal 17

This case is suitable for this algorithm , Karel used 5 **Beepers** and 31 **Steps** .

Other case :

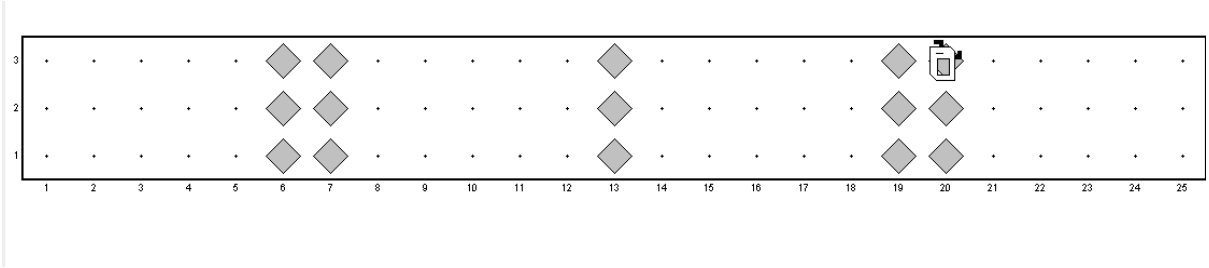



```
Row = 1
Column = 10
steps = 25
beepers = 6
```

In this case the Row is odd equal 1 , and Column is even equal 10

This case is suitable for this algorithm , Karel used 6 **Beepers** and 25 **Steps** .

I added this case when one of the dimensions is equal to 3 and the other is greater than 15. Here in THE Drow Plus (General Algorithm), a large number of Beepers and more steps will be used. Here is an example of the case:



```
Row = 3
Column = 25
steps = 72
beepers = 15
```

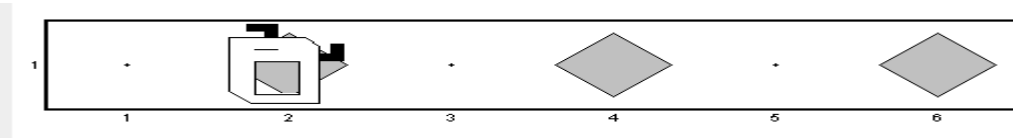
In this case the Row is odd equal 3 , and Column is odd equal 25

This case is suitable for this algorithm , Karel used 16 **Beepers** and 72 **Steps** .

In the previous way it needs to 27 **Beepers** and more **Steps** .

The special case in this algorithm is that one of the dimensions is less than 7, and here the map cannot be divided into four sections.

We will divide them according to their dimensions, as in the following cases:

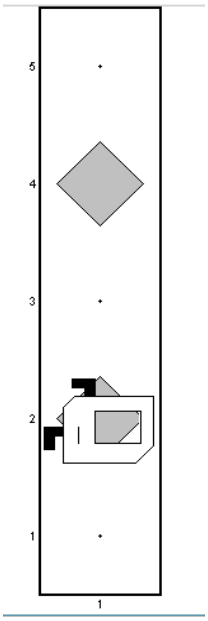



```
Row = 1
Column = 6
steps = 10
beepers = 3
```

In this case the Row is odd equal 1 , and Column is even equal 6

This case is suitable for this algorithm , Karel used 3 **Beepers** and 10 **Steps** , Here it is divided into **three** .

Other case :

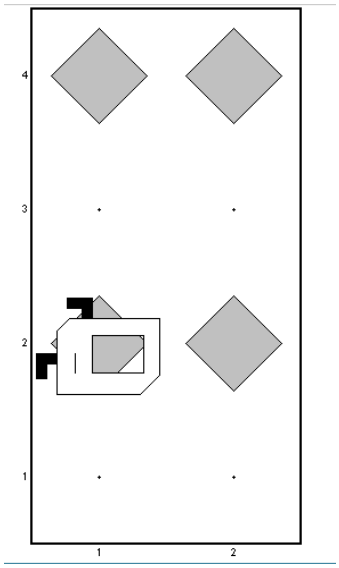



```
Row = 5
Column = 1
steps = 8
beepers = 2
```

In this case the Row is odd equal 5 , and Column is odd equal 1

This case is suitable for this algorithm , Karel used 2 **Beepers** and 8 **Steps** , Here it is divided into **two** .

Other case :

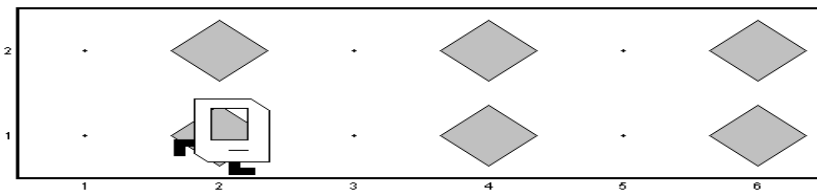


```
Row = 4
Column = 2
steps = 10
beepers = 4
```

In this case the Row is even equal 4 , and Column is even equal 2

This case is suitable for this algorithm , Karel used 4 **Beepers** and 10 **Steps** , Here it is divided into **two** .

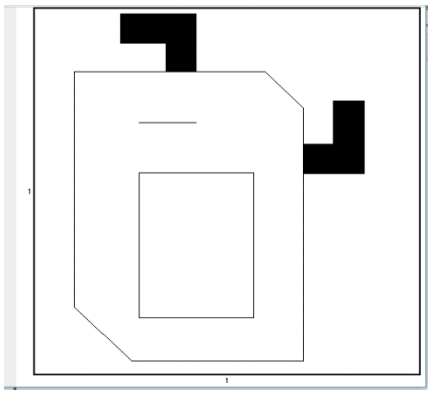
Other case :

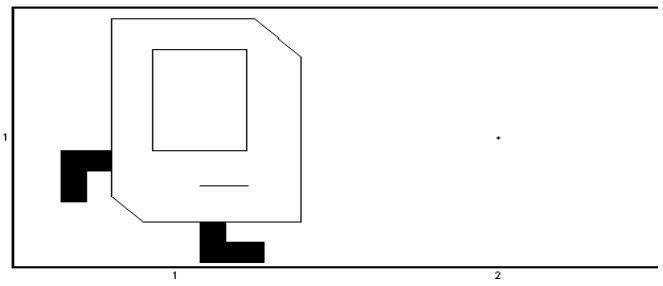


```
Row = 2
Column = 6
steps = 14
beepers = 6
```

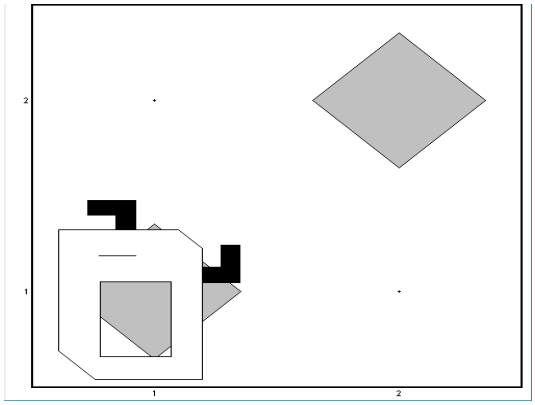
Special Cases :

It can't be divided when it is 1*1 OR 2*1





Other Special Cases :



```
Row = 2
Column = 2
steps = 5
beepers = 2
```

In this case the Row is even equal 2 , and Column is even equal 2

This case is suitable for this algorithm , Karel used 2 **Beepers** and 5 **Steps** , Here it is divided into **two** .

Here the algorithm was used to **Draw Diagonally Algorithm** but only one diameter was drawn .

In the end, I think I included all the special and natural cases.

I will put the video link here: (10) Karel Assignment - YouTube